Orquestração de Containers com Kubernetes usando KIND, Lens e Kubernetes Dashboard

Este projeto demonstra a criação de um cluster Kubernetes local utilizando o KIND, além da instalação de ferramentas visuais como o Kubernetes Dashboard e Lens para visualização da orquestração de containers e métricas de cada serviço.

☆ Introdução ao Kubernetes

O que é Kubernetes?

Kubernetes (também conhecido como K8s) é uma plataforma open-source para orquestração de containers que automatiza a implantação, escalonamento e gerenciamento de aplicações em containers. Ele foi originalmente desenvolvido pelo Google e agora é mantido pela Cloud Native Computing Foundation (CNCF).

O Kubernetes funciona como um sistema operacional para aplicações em containers, sendo responsável por gerenciar onde e como essas aplicações rodam em um cluster (um conjunto de máquinas chamadas de nodes).

◈ Nodes

Um node é uma máquina (física ou virtual) que faz parte do cluster Kubernetes. Existem dois tipos:

1 - Node de controle (Control Plane): é o cérebro do Kubernetes. Ele decide quando e onde os containers vão rodar, monitora o estado do cluster, e reage a falhas que podem acontecer.

2 - Node de trabalho (Worker Node): é onde os containers realmente rodam. Cada node de trabalho possui:

- Kubelet: agente responsável por comunicar o node com o control plane.
- Kube-Proxy: gerencia o tráfego de rede dentro e fora do node.
- Container Runtime (ex: Docker, containerd): roda os containers.

◈ Pods

O Pod é a menor unidade que pode ser implantada no Kubernetes. Ele encapsula um ou mais containers que compartilham:

- O mesmo endereço IP
- Espaço de armazenamento
- Configurações de rede

Mesmo que, na maioria dos casos, cada pod contenha apenas um container, é possível que múltiplos containers rodem juntos dentro do mesmo pod, geralmente quando precisam se comunicar com muita

frequência.

Como tudo se conecta?

- O usuário ou sistema envia uma instrução (ex: criar uma aplicação).
- O Control Plane processa e planeja onde a aplicação deve rodar.
- Um Pod é criado em um Worker Node com os containers desejados.
- O Kubelet no node gerencia o pod e garante que ele esteja sempre funcionando.
- Se o pod falhar, o Kubernetes reinicia automaticamente.

🎯 Objetivo do Kubernetes

- **Automatizar** a implantação, escalonamento e gerenciamento de aplicações em containers
- Garantir **alta disponibilidade** (sem nenhum downtime)
- Gerenciar **recursos de forma eficiente** (CPU, memória, armazenamento)
- Facilitar **descoberta de serviços** e balanceamento de carga
- Permitir **atualizações contínuas** e rollbacks

Problemas que ele resolve:

- Containers que falharem são reiniciados automaticamente
- Escalonamento rápido de aplicações para evitar sobrecarga
- Atualizações são aplicadas sem causarem o "Downtime"

🔧 Tecnologias utilizadas

- Kubernetes
- KIND (Kubernetes IN Docker)
- Lens
- Kubernetes Dashboard

📦 Pré-requisitos e Instalação

Ubuntu/Debian

```
# Instalar Docker
sudo apt update
sudo apt install -y ca-certificates curl gnupg lsb-release

sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
  sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

```
echo \
  "deb [arch=$(dpkg --print-architecture) \
  signed-by=/etc/apt/keyrings/docker.gpg] \
  https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt update

sudo usermod -aG docker $USER
newgrp docker


# Instalar KIND
sudo apt install curl
snap install kubectl --classic
curl -Lo ./kind "https://kind.sigs.k8s.io/dl/v0.22.0/kind-linux-amd64"
sudo mv ./kind /usr/local/bin/kind
chmod +x /usr/local/bin/kind

# Ativar o Docker
sudo systemctl start docker
sudo systemctl enable docker
```

Fedora

```
# Instalar o Docker
sudo dnf install docker kubectl git -y
sudo usermod -aG docker $USER
newgrp docker

# Instalar KIND
curl -Lo ./kind "https://kind.sigs.k8s.io/dl/v0.22.0/kind-linux-amd64"
sudo mv ./kind /usr/local/bin/kind
chmod +x /usr/local/bin/kind

# Ativar o Docker
sudo systemctl start docker
sudo systemctl enable docker
```

🚀 Passo a passo

1. Clone o repositório

```
git clone https://github.com/gabrielbariaguera/Kubernetes-kind.git
cd Kubernetes-kind
```

2. Acesso com Lens - Interface Gráfica (opcional)

2.1 Torne o script executável:

```
chmod +x lens-install.sh
```

2.2 Execute o script:

```
sudo ./lens-install.sh
```

Após a instalação execute o Lens.

3. Criação do cluster com KIND

```
kind create cluster --name NOME-CLUSTER --config kind-config.yaml
```

Abra o Lens e ele detectará o cluster automaticamente (para habilitar métricas vá para o passo 8).

4. Instalação do Kubernetes Dashboard - Interface Gráfica (obrigatória) e Metrics Server

```
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/dashboard/v2.7.0/aio/deploy/re
commended.yaml --validate=false
kubectl apply -f https://github.com/kubernetes-sigs/metrics-
server/releases/latest/download/components.yaml
```

5. Criação de Usuário e Geração do token para acesso ao Dashboard

Crie um usuário administrador utilizando o arquivo de configuração do repositório

```
kubectl apply -f dashboard-admin.yaml
```

Crie um token para o acesso ao dashboard e o copie:

```
kubectl -n kubernetes-dashboard create token admin-user
```

6. Usando o Port-Foward para o acesso ao Dashboard

```
kubectl -n kubernetes-dashboard port-forward svc/kubernetes-dashboard
8443:443
```

Acesse: https://localhost:8443 e entre com o token gerado anteriormente

7. Criando um Deployment Nginx para Demonstração de Pods (duas cópias idênticas)

No Kubernetes, containers são encapsulados em **Pods** (a menor unidade deployável).

Vamos criar um deployment:

```
kubectl create deployment nginx-dashboard --image=nginx:alpine --replicas=2
```

Expor o deployment como serviço:

```
kubectl expose deployment nginx-dashboard --port=80 --type=NodePort
```

Verificar os pods criados:

```
kubectl get pods -l app=nginx-dashboard -o wide
```

Os Nodes são máquinas virtuais (VMs) que fazem parte do cluster do Kubernetes, é como se fosse o servidor que deixa os Pods no ar

Para visualizar containers dentro de um Pod use:

```
kubectl describe pod NOME-DO-POD | grep -A 5 "Containers:"
```

8. Criando um Deployment de um simples site HTML

Vamos utilizar os arquivos disponibilizados no repositório (index.html, Dockerfile e k8s-deployment)

Vamos criar a imagem Docker:

```
docker build -t localhost/meu-site-nginx:latest .
```

Carregue a imagem em um cluster existente:

```

```

```
kind load docker-image meu-site-nginx --name NOME_DO_CLUSTER
```

Aplique as configurações do deployment:

```
kubectl apply -f k8s-deployment.yaml
kubectl apply -f site-service.yaml
```

Verifique se o serviço está rodando, caso esteja aplique um redirecionamento de porta:

```
kubectl port-forward service/meu-site 8080:80
```

Agora, acesse no seu navegador:

http://localhost:8080

---

9. Exemplos de Orquestração do Kubernetes:

9.1 Escalabilidade:

Escalabilidade: criando réplicas dos Pods já existentes

```
kubectl scale deployment meu-site --replicas=5
```

Utilize para ver atualizações em tempo real:

```
watch -n 1 kubectl get pods
```

O Kubernetes permite a escalabilidade em tempo real sem Downtime

9.2 Escalabilidade Automática:

O Kubernetes consegue escalonar a aplicação automaticamente com o comando:

```
kubectl autoscale deployment meu-site --min=1 --max=5 --cpu-percent=50
```

Nesse comando é definido o deployment (meu-site) a ser escalonado, o mínimo de réplicas de pods (1), o máximo de réplicas de pods (5) e quando o escalonamento deve ser feito (50% cpu)

9.3 Limitação de Recursos

É possível limitar os recursos utilizados por cada container no Kubernetes

Aplicar limitação de cpu e memória com comando direto:

```
kubectl patch deployment meu-site --patch '
spec:
  template:
    spec:
      containers:
      - name: nginx  # Coloque o nome do seu container
        resources:
          requests:
            cpu: "100m"
            memory: "256Mi"
          limits:
            cpu: "500m"
            memory: "512Mi"
'
```

Esse comando define os limites (limits) e os recursos garantidos pelo container (requests)

Para ver os limites aplicados use:

```
kubectl describe pod NOME_DO_POD | grep -A 5 "Limits"
```

9.4 Exemplo de Auto-Recuperação:

Liste todos os Pods:

```
kubectl get pods
```

Escolha um e, de maneira forçada, remova um deles:

```
kubectl delete pod NOME-POD-ESCOLHIDO --force
```

O Kubernetes automaticamente cria um novo Pod para substituir o deletado/com erro

9.5 Rollback

Verifique o histórico de atualizações:

```
kubectl rollout history deployment/meu-site
```

Volte uma versão anterior, é como dar um "CTRL Z" na sua aplicação!

```
kubectl rollout undo deployment/meu-site
```

Ou até mesmo específique uma versão específica (voltando a versão 1):

```
kubectl rollout undo deployment/meu-site --to-revision=1
```

O Kubernetes consegue fazer essas trocas de versões sem interromper a aplicação!

Atualizando a imagem para uma versão inexistente para simular erros:

```
kubectl set image deployment/meu-site nginx=nginx:versao-inexistente
```

Dê uma olhada nos pods falhando, e então volte para a versão anterior e o Kubernetes consegue recuperá-los automaticamente:

```
watch kubectl get pods
```

📎 Referências

Documentação Kubernetes
Documentação Kind
Documentação Docker